

Approximating General Metric Distances Between a Pattern and a Text

Ely Porat*

Klim Efremenko^{†‡}

February 11, 2008

Abstract

Let $T = t_0 \dots t_{n-1}$ be a text and $P = p_0 \dots p_{m-1}$ a pattern taken from some finite alphabet set Σ , and let \mathbf{d} be a metric on Σ . We consider the problem of calculating the sum of distances between the symbols of P and the symbols of substrings of T of length m for all possible offsets. We present an ε -approximation algorithm for this problem which runs in time $O(\frac{1}{\varepsilon^2} n \cdot \text{polylog}(n, |\Sigma|))$.

1 Introduction

String matching, the problem of finding all occurrences of a given pattern in a given text, is a classical problem in computer science. The problem has pleasing theoretical features and a number of direct applications to “real world” problems.

Advances in multimedia, digital libraries, and computational biology, have shown that a much more generalized theoretical basis of string matching could be of tremendous benefit [?, ?]. To this end, string matching has had to adapt itself to increasingly broader definitions of “matching”. Two types of problems need to be addressed – *generalized matching* and *approximate matching*. In generalized matching, one seeks all exact occurrences of the pattern in the text, but the “matching” relation is defined differently. The output is all locations in the text where the pattern “matches” according to the new definition of a match. The different applications define the matching relation. Examples can be seen in Baker’s *parameterized matching* ([?]) or Amir and Farach’s *less-than matching* ([?]). The second model, and the one we are concerned with in this paper, is that of *approximate matching*. In approximate matching, one defines a distance metric between the objects (e.g. strings, matrices) and seeks to calculate this distance for all text locations. Usually we seek locations where this distance is small enough.

One of the earliest and most natural metrics is the *Hamming distance*, where the distance between two strings is the number of mismatching characters exists algorithm calculating

*Department of Computer Science, Bar-Ilan University, 52900 Ramat-Gan, Israel; porately@cs.biu.ac.il

[†]Department of Computer Science, Bar-Ilan University, 52900 Ramat-Gan, Israel

[‡]Weizmann Institute, Rehovot 76100, Israel

this distance exactly [?] and approximating it [?]. Levenshtein [?] identified three types of errors: mismatches, insertions, and deletions. These operations are traditionally used to define the *edit distance* between two strings. The edit distance is the *minimum* number of edit operations one needs to perform on the pattern in order to achieve an exact match at the given text location. Lowrance and Wagner [?, ?] added the *swap* operation to the set of operations defining the distance metric. Much of the recent research in string matching concerns itself with understanding the inherent “hardness” of the various distance metrics, by seeking upper and lower bounds for string matching under these conditions.

A natural subset of these problems is when the distance is defined only on the alphabet, and therefore, the distance between two strings is the sum of the distances between the corresponding characters in both strings. It is possible to solve this problem in time $O(n \cdot m)$ by employing the naive approach of summing the distances between each character of the pattern, and its corresponding character in the text, for each possible alignment of the pattern. This problem was first defined by Muthukrishnan in [?] and has been open since. In this paper we present an approximation algorithm for this problem.

This algorithm consists of two parts: the first part is a preprocessing phase in which random hash functions on the alphabet is constructed. We use same hashing which Bartal used for tree embedding at [?]. We use this hashing in order to separate the places where distance between letters is large and places where this distance is small.

The second part of the algorithm is an application of sampling ([?]), which allows us to give an approximation of the distance between the text and the pattern, in time $O(\frac{1}{\epsilon^2} n \cdot \text{polylog}(n, |\Sigma|))$.

The contributions of this paper are twofold: on the technical side, we have solved a problem that has been *open* for over a decade, by presenting the *fastest* known approximation algorithm for many metrics; additionally, and this is perhaps the more important contribution of this paper, we have identified and exploited a new technique – *sampling*, that has been used in some recent papers ([?]) only implicitly. We employ sampling in a much more sophisticated manner and show how to use this important tool for approximating distances. We also present a novel way of using embeddings and geometry tools in pattern matching. This technique possesses a wide range of applications. For example, one can easily extend it to calculate the ℓ_2 -norm distance (in other words, when

$$\mathbf{Dist}(T[i \dots i + m - 1], P) = \sqrt{\sum_{j=0}^{m-1} \mathbf{d}(t_{i+j}, p_j)^2} \quad (1)$$

is the distance measure), or it can be extended for many infinite metrics. Our algorithm also allows for symbols in a text to be wildcards.

We believe that this new method for solving approximate string matching problems – embedding metric in some suitable space and sampling – may actually yield efficient algorithms for many more problems in the future.

2 Problem definition and Preliminaries

Definition 2.1. A metric space is a pair (X, \mathbf{d}) where X is a set of points and $\mathbf{d} : X \times X \rightarrow \mathbf{R}^+$ is a metric satisfying the following axioms:

$$\begin{aligned} \mathbf{d}(x_1, x_2) &= \mathbf{d}(x_2, x_1). \\ \mathbf{d}(x_1, x_3) &\leq \mathbf{d}(x_1, x_2) + \mathbf{d}(x_2, x_3). \\ \mathbf{d}(x_1, x_2) &= 0 \Leftrightarrow x_1 = x_2. \end{aligned} \tag{2}$$

Let (Σ, \mathbf{d}) be a metric space, and $A = a_0 \dots a_{m-1}, B = b_0 \dots b_{m-1}$ be any two strings of the same length with symbols from Σ . We define the *distance* of A and B by $\mathbf{Dist}(A, B) = \sum_{i=0}^{m-1} \mathbf{d}(a_i, b_i)$. Given a text $T = t_0 \dots t_{n-1}$ and a pattern $P = p_0 \dots p_{m-1}$, our goal is to calculate the array $S[i] = \mathbf{Dist}(T[i \dots i + m - 1], P)$, for each possible offset $i = 0, \dots, n - m - 1$. Calculating the exact values of S can be done in $O(nm)$ time, using the naive approach, In most cases it is enough to know only an approximation of the distance; we therefore present an efficient algorithm which approximates the values of S .

Convolutions can be used in the standard fashion to improve the time for finite fixed alphabets.

Definition 2.2. Let $A[0], \dots, A[n - 1]$ and $B[0], \dots, B[m - 1]$ be arrays of natural numbers. The discrete convolution (polynomial multiplication) of A and B is V where:

$$V[i] = \sum_{j=0}^{m-1} A[i - j]B[j], \tag{3}$$

where $i = 0, \dots, n - m$. We denote V as $A * B$. We will choose $A = T$ and $B = P$, i.e. we will treat T and P as the coefficients of polynomials of degrees $n - 1$ and $m - 1$, respectively.

By standard tricks, namely, (1) reversing the text to obtain T^R ; (2) calculating $T^R * P$; (3) reverse the result; (4) discard the first $m - 1$ values, and last $n - m + 1$ values of the result, we obtain an array $V = (T^R * P)^R[m - 1 \dots n - 1]$ where for each i ,

$$V[i] = \sum_{j=0}^{m-1} t_{i+j}p_j. \tag{4}$$

In other words, for every possible offset i , $V[i]$ is the sum of the pattern symbols multiplied, each with its corresponding text symbol. For convenience, we define $T \otimes P = (T^R * P)^R[m - 1 \dots n - 1]$

A convolution can be computed in time $O(n \log m)$, in a computational model with word size $O(\log m)$, by using the Fast Fourier Transform (FFT) [?].

Remark 2.3. Using FFT we can compute general pattern distances, in time $\mathcal{O}(|\Sigma| n \log m)$, by using the following method: for every $a \in \Sigma$, we define an array $\chi_a(P)$ by setting $\chi_a(P)[i] = \chi_a(P[i])$, where $\chi_a(x) = 1$ if $x = a$ and 0 otherwise. Set $T_a[i] = \mathbf{d}(a, t_i)$. Computing $T_a \otimes \chi_a(P)$ gives us the sum of the distances of the letter a from the text. The sum of all convolution results is the desired distance.

We provide some required definitions regarding metric spaces:

Definition 2.4. Let ρ be a mapping $\rho: X \rightarrow Y$, where (X, \mathbf{d}_1) and (Y, \mathbf{d}_2) are metric spaces. ρ is an isometry if $\forall x, y \in X \mathbf{d}_1(x, y) = \mathbf{d}_2(\rho(x), \rho(y))$.

Unfortunately, we cannot always construct an isometry. Therefore we consider weaker conditions:

Definition 2.5. Given two metric spaces (X, \mathbf{d}_1) and (Y, \mathbf{d}_2) , and a value $c \geq 1$, a mapping $\rho: X \rightarrow Y$ is called a c -distortion embedding, if for all $x, y \in X$,

$$\mathbf{d}_1(x, y) \leq \mathbf{d}_2(\rho(x), \rho(y)) \leq c \cdot \mathbf{d}_1(x, y). \quad (5)$$

3 The One-Mismatch Algorithm

In this section we describe the *one-mismatch* algorithm, in itself a very useful general tool in pattern matching. The one-mismatch algorithm had been described before in [?, ?].

Given a numeric text T and a numeric pattern P , and we want to find exact matches of P in T . One way to do so is by calculating for each location $0 \leq i \leq n - m - 1$ the value:

$$\sum_{j=0}^{m-1} (p_j - t_{i+j})^2 = \sum_{j=0}^m (p_j^2 - 2p_j t_{i+j} + t_{i+j}^2) .$$

Notice this sum will be zero iff there is an exact match at location i . Furthermore this sum can be computed efficiently for all i 's in $\mathcal{O}(n \log m)$ time using convolutions. Notice that if P and T are not numeric then an arbitrary one-to-one mapping can be chosen from the alphabet to the set of positive integers N .

This method can be extended to the case of matching with “don't cares” [?], by simply calculating instead

$$A_0[i] = \sum_{j=0}^{m-1} p'_j t'_{i+j} (p_j - t_{i+j})^2 ,$$

where $p'_j = 0$ (resp. $t'_j = 0$) if p_j (resp. t_j) is a “don't care” symbol and 1 otherwise. Wherever there is an exact match with “don't cares” this sum will be exactly 0. This can also be computed with convolutions in time $\mathcal{O}(n \log m)$.

Again, this scheme can be further extended to the one-mismatch problem, which is to determine if P matches $T[i..i + m - 1]$ with at most one mismatch. Furthermore, we can identify the location of the mismatch for each such i . This is done by also computing, for each i ,

$$A_1[i] = \sum_{j=0}^{m-1} j p'_j t'_{i+j} (p_j - t_{i+j})^2 ,$$

by using the convolution. Then if p matches the text at offset i with one mismatch then eventually $A_0[i] = (p_r - t_{i+r})^2$ and $A_1[i] = r(p_r - t_{i+r})^2$ where r is a location of a mismatch. Therefore, by calculating $A_1[i]/A_0[i]$, we find the supposed mismatch location and verify

it. Finally, locations where exact matches occur will be labeled “match”, location where a single mismatch occurs will be labeled with its location, and location where more than one mismatch occurs will be labeled \perp . The one-mismatch algorithm is therefore as follows:

1. compute the array

$$A_0[i] = \sum_{j=0}^m p'_j t'_{i+j} (p_j - t_{i+j})^2$$

using FFT.

2. compute the array

$$A_1[i] = \sum_{j=0}^m j p'_j t'_{i+j} (p_j - t_{i+j})^2$$

using FFT.

3. If $A_0[i] = 0$ set $B[i] \leftarrow$ “match” else set $B[i] \leftarrow A_1[i]/A_0[i]$.
4. For each i s.t. $B[i] \neq$ “match”, check to see if $(p[B[i]] - t[B[i] + i])^2 = A_0[i]$. If this is not the case then set $B[i] \leftarrow \perp$.

The running time of this algorithm is $\mathcal{O}(n \log m)$.

4 The Sampling Method

In this section we present a general method referred to as the *sampling* method. It allows us, for every possible offset, to sample (i.e. choose) a *random mismatch* from the set of all mismatches w.r.t. this offset. We show how to utilize the previously described algorithm for this purpose.

First fix some probability $0 < q \leq 1$, and define subpattern P^* of P by:

$$p_i^* = \begin{cases} p_i & \text{with probability } q; \\ \phi(\text{don't care}) & \text{otherwise.} \end{cases} \quad (6)$$

In the algorithm referred to as $\text{Sample}(q, T, P)$, we simply create P^* as defined in (6) and run the one-mismatch algorithm on P^* and T . Now, for every offset i , let m_i be the number of mismatches between T and P w.r.t. this offset. The following lemma trivially follows:

Lemma 4.1. *Let B be the array returned by $\text{Sample}(q, T, P)$. For some location i ,*

$$\Pr(B[i] = \text{“match”}) = (1 - q)^{m_i} ,$$

and

$$\Pr(B[i] \text{ is a mismatch location}) = m_i q (1 - q)^{m_i - 1} .$$

Another important property of this algorithm is that the mismatch returned w.r.t. offset i is uniformly distributed over the set of all mismatches w.r.t. offset i . We will show how to use this algorithm to sample a random location for which the distance is not 0. Notice that for $q \approx \frac{1}{m_i}$, the probability of finding a mismatch w.r.t. offset i is $\mathcal{O}(1)$. Therefore, we can enumerate on $q = \{2^{-j}\}_{j=0}^{\log m}$. Then, for every location i there exists some q which is $\approx \frac{1}{m_i}$. Therefore, the next algorithm finds for each location a mismatch with constant probability, which is uniformly distributed over all the mismatches.

1. **for** $q = 1; q \geq 1/m ; q = q/2$
2. Sample(q, T, P)
3. For every offset i if a mismatch is found return it.

5 Motivation for the Algorithm

Remark 5.1. *In this paper we assume that the ratio of maximal and minimal distances is bounded by B_d . Therefore, w.l.o.g. we can assume that the minimal nonzero distance is 1, i.e. $\forall x, y \in \Sigma, \mathbf{d}(x, y) \leq B_d$ and $\mathbf{d}(x, y) > 0 \iff \mathbf{d}(x, y) \geq 1$. That is because if B_{\min} is the minimal distance, then we can use the metric $\frac{d}{B_{\min}}$ instead.*

A first naive approach to approximate the distance is as follows: say we wish to provide an approximation only for some offset i , and let X be a random variable which is equal to $\mathbf{d}(t_{i+j}, p_j)$, where j is chosen uniformly from $0, 1, \dots, m-1$. We can sample X by choosing a random j and calculating $\mathbf{d}(t_{i+j}, p_j)$. The expectation of $m \cdot X$ is the desired sum. Therefore, the way to compute $\mathbf{E}(X)$ is sample X several times and return the average. The problem with this approach is that the variance of X may be very large: for example, if P matches T except for a few mismatches, then w.h.p. we will not sample even a single mismatch.

The second attempt to reduce the variance of variable X , is to use the sampling algorithm described in Sect. 4. As a result, X will be distributed only over locations where $\mathbf{d}(t_{i+j}, p_j) > 0$. That is because the sampling algorithm returns only relative locations j for which $t_{i+j} \neq p_j$. This sampling approach reduces the variance of X , but still it may happen that for some offset i , all distances are very small except for a single one which is even greater than the sum of all others. With high probability, only the smaller distances will be sampled, thus affecting the final outcome. This approach can provide us with an algorithm which runs in $O(n \cdot B_d \cdot \text{polylog}(n, |\Sigma|))$ time, however, B_d may be very large.

All the above leads us to search for a way to sample only locations j for which, when some D is fixed, $D \leq \mathbf{d}(t_{i+j}, p_j) < 2D$. Then, with an additional multiplicative factor of $\log(B_d)$, we can enumerate on $D = \{2^i\}_{i=0}^{\log B_d}$, each step approximating the expectation of the variable X_D which uniformly ranges over $\{\mathbf{d}(t_{i+j}, p_j) \mid D \leq \mathbf{d}(t_{i+j}, p_j) < 2D\}$. Notice that for any value a ,

$$\Pr(X_D = a) = \begin{cases} \frac{\Pr(X=a)}{\Pr(D \leq X < 2D)} & D \leq a < 2D, \\ 0 & \text{else.} \end{cases}$$

Hence it follows that

$$\mathbf{E}(X) = \sum_D \Pr(D \leq X < 2D) \mathbf{E}(X_D) .$$

A hypothetic way to sample X_D would be to design a mapping π_D on Σ for which

$$D < \mathbf{d}(x, y) < 2D \iff \pi_D(x) \neq \pi_D(y) . \quad (7)$$

If so, we could have run the sampling algorithm on $\pi_{\mathbf{d}}(T)$ and $\pi_{\mathbf{d}}(P)$, applying $\pi_{\mathbf{d}}$ in the obvious way, and obtain samples of X_D . Then, the average of sampled distances will approximate $\mathbf{E}(X_D)$. The approximation of $\Pr(D \leq X < 2D)$ would have been also simple: it is a number of approximated mismatches divided by m (i.e. the length of the pattern). Unfortunately, we cannot design such a mapping. However, we can design a set of mappings such that for a random mapping, this condition holds with high probability.

6 Probabilistically Separating Hashing

In this section, our goal is to construct a random hash π for a given D such that (7) holds with good probability. The set of hash functions \mathcal{H}_D called *C-Probabilistically Separating Hashing* if it admits next two conditions:

1. If the distance between x, y is greater than D , then they their hashing is different i.e.
 $\mathbf{d}(x, y) \geq D \Rightarrow \forall \pi \in \mathcal{H}_D \pi(x) \neq \pi(y)$.
2. $\forall x, y \in \Sigma, \Pr_{\pi \in \mathcal{H}_D}(\pi(x) \neq \pi(y)) \leq C \frac{\mathbf{d}(x, y)}{D}$.

Bartal at [?] gave a construction of $\log |\Sigma|$ -Probabilistically Separating Hashing for finite metrics after it was extended for graphs embedded in real normed spaces at [?]. In section 8 we will give a simple construction for the case when alphabet is normed space \mathbb{R}^d with small d .

Notice that we only need to build such a hashing only once for every alphabet. Therefore it can be done as a preprocessing measure.

We are able to use π_D in order to sample a subset of indices for which the distance is not too small. We will now show that this will also allow us to sample X_D as we desire.

Lemma 6.1. *Fix an offset i . Let $A = \{j \mid \pi_D(t_{i+j}) \neq \pi_D(p_j)\}$ be the set of mismatches under π_D and $B = \{j \mid D < \mathbf{d}(t_{i+j}, p_j) \leq 2D\}$ be the set of indices we are really interested in sampling from them. Then:*

1. $\mathbf{E}_{\pi_D \in \mathcal{H}_D}(|A|) = \mathcal{O}(\frac{S \cdot C}{D})$ (where the expectation is over the choice of π_D)
2. $B \subseteq A$ and $\frac{S_D}{D} \leq |B| \leq \frac{2S_D}{D}$.

Where $S = \sum_{j=0}^{m-1} \mathbf{d}(p_j, t_{i+j})$ and $S_D = \sum_{j \in B} \mathbf{d}(p_j, t_{i+j})$.

Proof. By linearity of expectation:

$$\mathbf{E}(|A|) = \sum_j \Pr(\pi_D(t_{i+j}) \neq \pi_D(p_j)).$$

By definition of Probabilistically Separating Hashing we have:

$$\sum_j \Pr(\pi_D(t_{i+j}) \neq \pi_D(p_j)) \leq \sum_j \frac{C \mathbf{d}(p_j, t_{i+j})}{D} = \frac{C \cdot S}{D}.$$

This proves (1).

$B \subseteq A$ follows from theorem 8.1 and $\frac{S_D}{D} = \sum_{j \in B} \frac{\mathbf{d}(p_j, t_{i+j})}{D}$ but $1 \leq \frac{\mathbf{d}(p_j, t_{i+j})}{D} \leq 2$ for $j \in B$ so $\frac{S_D}{D} \leq |B| \leq \frac{2S_D}{D}$ \square

7 The Algorithm

At this point, we have all the tools necessary in order to describe the algorithm. The algorithm is based upon the application of sampling algorithm, described previously, to the C -probabilistically separating hash provided in Sect. 6. As a preprocessing phase, we construct for the metric space (Σ, \mathbf{d}) , samples of hashing $\pi_D \in \mathcal{H}_D$ for $D = 2^i$. The preprocessing algorithm therefore gets a metric space (Σ, \mathbf{d}) , where Σ is the alphabet and \mathbf{d} is the metric on it, and produces the $\mathcal{O}(\frac{1}{\epsilon^2} \log |\Sigma| \log m)$ hash functions π_D chosen at random from \mathcal{H}_D .

The main (i.e. query) algorithm gets a text $T = t_0 t_1 \dots t_{n-1}$ and a pattern $P = p_0 \dots p_{m-1}$ over the alphabet Σ . For a fixed offset i the result will be in $((1 - \epsilon)S_i, (1 + \epsilon)S_i)$ with probability $1 - e^{-t}$. The output of the algorithm is an array $R[0 \dots n - m - 1]$ where $R[i]$ is an ϵ -approximation to $S[i] = \sum_{j=0}^{m-1} \mathbf{d}(p_j, t_{i+j})$ i.e:

$$\forall i \quad \Pr(|P[i] - S[i]| \geq \epsilon S[i]) \leq e^{-t} \quad (8)$$

We will now outline the idea of the algorithm. We want to approximate

$$m \mathbf{E}(X) = \sum_D m \Pr(D \leq X < 2D) \mathbf{E}(X_D).$$

We will enumerate D , increasing it each time by a factor of 2, and approximate $m \Pr(D \leq X < 2D) \mathbf{E}(X_D)$. Fix some D and some offset i , let as before $A = \{j \mid \pi_D(t_{i+j}) \neq \pi_D(p_j)\}$, where A depends on the random mapping π_D , and $B = \{j \mid D < \mathbf{d}(t_{i+j}, p_j) \leq 2D\}$. Recall that $B \subseteq A$, and that $\frac{|B|}{|A|}$ is not too small.

In order to approximate $\mathbf{E}(X_D)$ we will use the sampling algorithm on $\pi_D(P)$ and $\pi_D(T)$. We get a random element in A , and we check if this element is also in B . In order to approximate $\mathbf{E}(X_D)$ we average the distances of elements found in B .

In order to approximate $m \Pr(D \leq X < 2D) = |B|$ we use lemma 4.1. The probability that the sampling algorithm returns “match” is $q_0 = \mathbf{E}(1 - q)^{|A|}$, and the probability that it returns a mismatch from the set B is $q_1 = |B| q \mathbf{E}(1 - q)^{|A|-1}$. So, $|B| = \frac{q_1(1-q)}{qq_0}$.

Let's assume that we run the sampling algorithm K times; then the total number of matches is $m_0 \approx Kq_0$ and the total number of elements in B is $m_1 \approx Kq_1$. Let M_1 be an array of the elements in B which were found, including repetitions of elements from B . $|M_1| = m_1$.

We will approximate $|B|$ by $\frac{m_1(1-q)}{qm_0}$ because:

$$|B| = \frac{\mathbf{E}(m_1)(1-q)}{q\mathbf{E}(m_0)}, \quad (9)$$

and approximate $\mathbf{E}(X_D)$ by $\frac{\sum_{j \in M_1} \mathbf{d}(p_j, t_{i+j})}{m_1}$. Therefore:

$$m \Pr(D \leq X < 2D) \mathbf{E}(X_D) \approx \frac{(1-q)}{qm_0} \sum_{j \in M_1} \mathbf{d}(p_j, t_{i+j}) \quad (10)$$

We will need to show that this approximation is narrow, i.e. that the variance of the approximation is small. In order to do so, we will need to choose q s.t. $q \approx \frac{1}{\mathbf{E}[A]}$. In order to find such a q , we try a series of q 's, increasing by a factor of 2 each time, and choose q s.t. m_0 is large enough and qm_0 is maximal. We prove that this produces a good q w.h.p.

We now write the complete algorithm. Set $K = \mathcal{O}(\frac{1}{\varepsilon^2} \cdot C \cdot t)$.

```

1: for  $D = B_d ; D \geq 1 ; D = D/2$  do
2:   for  $q = 1/2 ; q > \frac{1}{m} ; q = q/2$  do
3:     for  $iter = 1 ; iter \leq K ; iter = iter + 1$  do
4:       Choose a random  $\pi \in \mathcal{H}_D$ 
5:       run  $\text{Sample}(q, \pi(T), \pi(P))$ . Save the result as the  $iter$ -th result for this  $q$ .
6:     end for
7:   end for
8:   for all offset in text  $i$  do
9:     Calculate  $m_0(i, q)$  for all  $q$ 's - the number of matches
10:    Among all  $q$  such that  $m_0 \geq e^{-4} \cdot K$  choose  $q(i)$  s.t.  $q(i)m_0(i, q(i))$  is maximal
11:    Set  $M_1$  to be the set of distances between  $D$  and  $2D$  for this  $q$ .
12:    Calculate

$$S_D(i) = \frac{(1-q)}{qm_0} \sum_{j \in M_1} \mathbf{d}(t_{i+j}, p_j)$$

13:   end for
14: end for
15: for every offset  $i$  return  $R(i) = \sum_D S_D(i)$ 

```

Algorithm 1: General distance algorithm

The running time of this algorithm is: $\mathcal{O}(\frac{1}{\varepsilon^2} n \log^2 m \log |\Sigma| \log B_d)$. This is because the running time is mostly dominated by the Sampling function, which takes $\mathcal{O}(n \log m)$ time. The Sampling function is executed $\mathcal{O}(\frac{1}{\varepsilon^2} \log |\Sigma| \log m \log B_d)$ times.

Theorem 7.1. *For every offset i*

$$\Pr \left(\left| R(i) - \sum_{j=0}^{m-1} \mathbf{d}(p_j, t_{i+j}) \right| \geq \varepsilon R(i) \right) \leq e^{-t} , \quad (11)$$

or in other words our algorithm returns ε -approximation w.h.p.

The proof of this theorem appears in the appendix.

8 Explicit hashing constructions for normed spaces

Now in this section let us construct explicit d -Probabilistically Separating Hashing for normed space \mathcal{R}^d with $\mathcal{L}_p, 1 \leq p \leq \infty$ norm. The main problem with previous constructions [?] is that this hashing can't be calculated efficiently and usually it takes $\mathcal{O}(n^2)$ time to calculate one hash function. In case that points not given in advance this may be bottleneck of the algorithm. An other reason why this construction important is: if we have d -Probabilistically Separating Hashing for space X and we have embedding $f : Y \mapsto X$ with distortion c then we can construct cd -Probabilistically Separating Hashing for space Y . The problem of embedding metric spaces to real normed spaces where deeply investigated.

Our construction is the same for every norm \mathcal{L}_p . Let $\vec{\varepsilon}$ be a vector of d independent random variables with uniform distribution on $[0, 1]$. Define:

$$\pi_D(\vec{x}) = \left\lfloor \frac{\vec{x}}{D} - \vec{\varepsilon} \right\rfloor = \left(\left\lfloor \frac{x_1}{D} - \varepsilon_1 \right\rfloor, \dots, \left\lfloor \frac{x_d}{D} - \varepsilon_d \right\rfloor \right) . \quad (12)$$

Theorem 8.1. *The above mapping π_D satisfies the next properties:*

1. *If the distance between x, y is greater than $D \cdot d^{1/p}$, then their mapping is different i.e. $\forall x, y \in R^d, \|x - y\|_p \geq D \cdot d^{1/p} \Rightarrow \pi_D(x) \neq \pi_D(y)$.*
2. *$\forall x, y \in R^d, \Pr(\pi_D(x) \neq \pi_D(y)) \leq \frac{d \cdot \|x - y\|_p}{d^{1/p} D}$.*

Proof. As follows:

1. This is trivial:

$$\begin{aligned} Dd^{1/p} &\leq \|x - y\|_p \\ \Rightarrow \exists i |x_i - y_i| &\geq D \\ \Rightarrow \forall \vec{\varepsilon} \pi_D(x) &\neq \pi_D(y). \end{aligned}$$

2. If $\mathbf{d}(x, y) \geq \frac{D}{cd}$ then this inequality is trivial. Therefore assume $\mathbf{d}(x, y) \leq \frac{D}{cd}$:

$$\begin{aligned} \Pr(\pi_D(x) \neq \pi_D(y)) &\leq \\ \sum_{i=1}^d \mathbf{Prob}(\pi_D(x)_i \neq \pi_D(y)_i) &\leq \\ \sum_{i=1}^d \cdot \Pr\left(\lfloor \frac{x_i}{D} - \varepsilon_i \rfloor \neq \lfloor \frac{y_i}{D} - \varepsilon_i \rfloor\right) &\leq \\ \sum_{i=1}^d \frac{|x_i - y_i|}{D} = \frac{\|\vec{x} - \vec{y}\|_1}{D} &\leq \frac{d\|\vec{x} - \vec{y}\|_p}{d^{1/p}D} \end{aligned}$$

□

We choose to use π as our embedding, and notice π is easy to calculate, assuming we already have the c -embedding σ . This calculation can be done in $\mathcal{O}(d|\Sigma|)$ time.

Remark 8.2. Consider the set $\{\pi_D(x) \mid x \in \Sigma\}$. While each of its members is a vector of length d , when comparing these vectors we are only interested in checking equality. Therefore, in order to save space, we can replace each vector with a unique number in $\{1, \dots, |\Sigma|\}$.

9 Conclusions

We have presented the first non-trivial algorithm for the approximation of a large class of distances between text and pattern. We believe that the techniques we have presented here have a wide range of applications. A further interesting open question is to generalize these techniques to the case where the distance is not necessary a metric.

A The proof of the algorithm

Remark A.1. Here w.h.p. mean with probability more than $1 - e^{-t}$

We will now prove that the algorithm indeed approximated the distances for each i w.h.p. We will only sketch the proof.

Proof. (of Algorithm) Fix some offset i . Then for every D we set $B = \{j \mid D < \mathbf{d}(t_{i+j}, p_j) \leq 2D\}$ and $A = \{j \mid \pi_D(t_{i+j}) \neq \pi_D(p_j)\}$ two sets. Notice that $|A|$ is a random variable.

Claim A.1. W.h.p. for every D there exist $q(D)$ s.t. $m_0(q) \geq e^{-4}K$ and $q \cdot m_0 \geq \frac{e^{-4}K}{\mathbf{E}(|A|)}$

Proof. There exist q s.t. $\frac{1}{\mathbf{E}(|A|)} \leq q \leq \frac{2}{\mathbf{E}(|A|)}$. The probability of a match for this q is $q_0 = \mathbf{E}(1 - q)^{|A|}$ by Jensen's inequality $\mathbf{E}(1 - q)^{|A|} \geq (1 - q)^{\mathbf{E}(|A|)} \geq e^{-3}$. $m_0(q)$ have binomial distribution $B(q_0, K)$ and so w.h.p. $m_0 \geq e^{-4}K$. For this q it also holds that $q \cdot m_0 \geq \frac{e^{-4}K}{\mathbf{E}(|A|)}$. So for the q that the algorithm chose also holds that $q \cdot m_0 \geq \frac{e^{-4}K}{\mathbf{E}(|A|)}$. □

Claim A.2. *There exist a constant $\tilde{m}_0 = \mathbf{E}(m_0) = K \mathbf{E}(1 - q)^{|A|}$. Such that.*

$$(1 - \varepsilon)m_0(D) \leq \tilde{m}_0(D) \leq (1 + \varepsilon)m_0(D)$$

w.h.p.

Proof. This follows from the Chernoff bound. This is because m_0 is binomially distributed variable $B(K, p)$ with $p \geq e^{-4}$. \square

Lets $c_D = \frac{(1-q(D))D}{q(D)\tilde{m}_0(D)}$ be a constant and $\tilde{S}(i) = \sum c_D \sum_{j \in M_1(D)} \frac{\mathbf{d}(t_{i+j}, p_j)}{D}$

Claim A.3. *$\tilde{S}(i)$ is close to $R(i)$ w.h.p. i.e.*

$$(1 - \varepsilon)R(i) \leq \tilde{S}(i) \leq (1 + \varepsilon)R(i)$$

Proof. We can represent $R(i)$ as:

$$R(i) = \sum_D \sum_{j \in M_1(D)} \frac{(1 - q(D))D}{q(D)m_0(D)} \cdot \frac{\mathbf{d}(t_{i+j}, p_j)}{D}.$$

By the previous claim $m_0(D)$ is close to $\tilde{m}_0(D)$. \square

Claim A.4.

$$\mathbf{E}(\tilde{S}(i)) = \sum_{j=0}^{m-1} \mathbf{d}(p_i, t_{i+j}).$$

Proof.

$$\begin{aligned} \mathbf{E}(\tilde{S}(i)) &= \sum \frac{c_D}{D} \mathbf{E}(\sum_{j \in M_1(D)} \mathbf{d}(t_{i+j}, p_j)) = \\ &= \sum \frac{c_D}{D} \mathbf{E}|M_1(D)| \mathbf{E}(X_D), \end{aligned} \tag{13}$$

But we know that:

$$\begin{aligned} \frac{c_D \mathbf{E}(|M_1(D)|)}{D} &= \frac{(1 - q(D)) \mathbf{E}(m_1(D))}{q(D) \tilde{m}_0(D)} = \\ &= \frac{(1 - q(D)) \mathbf{E}(m_1(D))}{q(D) \mathbf{E}(m_0(D))}. \end{aligned} \tag{14}$$

By (9) we have that: $\frac{c_D \mathbf{E}(|M_1(D)|)}{D} = |B| = m \Pr(D \leq X_D < 2D)$. So we have that:

$$\begin{aligned} \mathbf{E}(\tilde{S}(i)) &= \sum_D m \Pr(D \leq X_D < 2D) \mathbf{E}(X_D) = \\ &= m \mathbf{E}(X) = \sum_{j=0}^{m-1} \mathbf{d}(p_i, t_{i+j}). \end{aligned} \tag{15}$$

\square

Claim A.5. *There exists a universal constant C s.t. w.h.p. :*

$$c_D \leq C \frac{\varepsilon^2}{t} \sum c_D |M_1(D)|$$

holds w.h.p.

Proof. The previous claim showed us that $S(i) \leq \sum 2c_D |M_1(D)|$ because $\frac{\mathbf{d}(p_i, t_{i+j})}{D} \leq 2$ for $j \in M_1(D)$. Therefore, it's enough to prove that $c_D \leq C \frac{\varepsilon^2}{t} S(i)$. By A.1 we know that $qm_0 \geq \frac{e^{-4K}}{\mathbf{E}(|A|)}$. So we have:

$$c_D = \frac{(1 - q(D))D}{q(D)\tilde{m}_0(D)} \leq \frac{C\mathbf{E}(|A|)D}{K}. \quad (16)$$

By 6.1 $\mathbf{E}(|A|)D = \mathcal{O}(S \cdot c \cdot d)$. $K = \mathcal{O}(\frac{1}{\varepsilon^2} \cdot c \cdot d \cdot t)$ So

$$\frac{C\mathbf{E}(|A|)D}{K} = \mathcal{O}(\frac{\varepsilon^2}{t} S) \quad (17)$$

□

We will state the following lemma without proving it because it follows from the Chernoff bound:

Lemma A.2. *There exists a universal constant C s.t. for every sequence of independent random variables $X_1, X_2 \dots X_n$ with $1 \leq X_i \leq 2$, and a sequence of positive constants c_1, c_2, \dots, c_n s.t. $c_i < C \frac{\varepsilon^2}{t} \sum_{i=1}^n c_i$. Then:*

$$\Pr \left(\left| \sum_{i=1}^n c_i \cdot X_i - \mathbf{E} \left(\sum_{i=1}^n c_i \cdot X_i \right) \right| \geq \varepsilon \mathbf{E} \left(\sum_{i=1}^n c_i \cdot X_i \right) \right) \leq e^{-t}$$

Claim A.6.

$$\Pr \left(\left| \tilde{S} - \sum_{j=0}^{m-1} \mathbf{d}(p_i, t_{i+j}) \right| \geq \varepsilon \tilde{S} \right) \leq e^{-t}. \quad (18)$$

Proof. $\tilde{S} = \sum c_i \frac{X_D}{D}$ by definition $1 \leq \frac{X_D}{D} \leq 2$, by A.4 $\mathbf{E}(\tilde{S}) = \sum_{j=0}^{m-1} \mathbf{d}(p_i, t_{i+j})$ by A.5 follows that $c_i < C \frac{\varepsilon^2}{t} \sum_{i=1}^n c_i$ and therefore A.2 proves the claim □

□